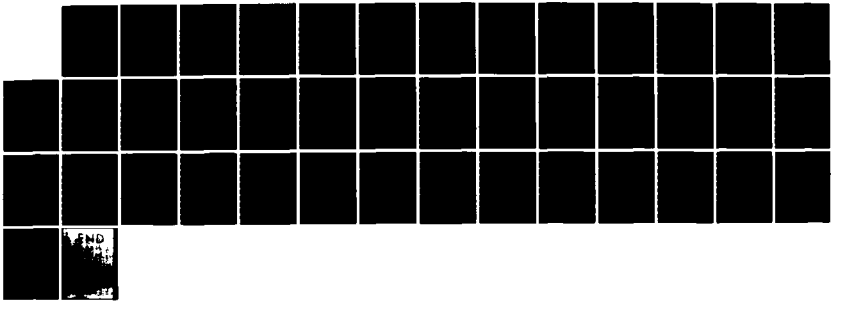
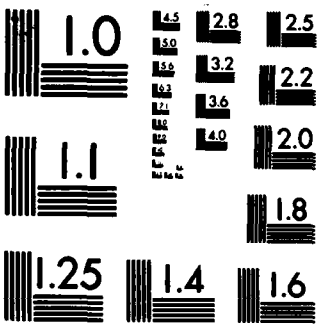
AD-A145 424 PARALLEL SEARCHING AND MERGING ON ZMOB(U) MARYLAND UNIV 1/1
COLLEGE PARK CENTER FOR AUTOMATION RESEARCH S KASIF
JUN 84 CAR-TR-64 AFOSR-TR-84-0746 F49620-83-C-0082
UNCLASSIFIED F/G 9/2 NL





4

CAR-TR-64
CSC-TR-1405

F49620-83-C-0082
June 1984

PARALLEL SEARCHING AND MERGING ON ZMOB

Simon Kasif
Center for Automation Research
University of Maryland
College Park, MD 20742

CENTER FOR AUTOMATION RESEARCH

UNIVERSITY OF MARYLAND
COLLEGE PARK, MARYLAND
20742

CAR-TR-64
CSC-TR-1405

F49620-83-C-0082
June 1984

PARALLEL SEARCHING AND MERGING ON ZMOB

Simon Kasif
Center for Automation Research
University of Maryland
College Park, MD 20742

ABSTRACT

One of the most difficult issues that must be addressed when studying a class of parallel algorithms is the problem of choosing a model that captures the inherent difficulty of implementing these algorithms on a multiprocessor architecture. Shared memory models have proven to be an effective tool for deriving lower bounds on the complexity of comparison problems. In particular, a speed-up of lg(P) is possible for the problem of finding an element in an N-element sorted list, and speed-ups of P/lglgP and P are possible for merging N-element sorted lists on P processors for the cases of N=P and P<N respectively.

In practice, these speed-ups are not attainable since the shared memory models ignore many practical considerations in multiprocessor systems, such as interprocessor communications, distribution of data on local memories and limited fan-out of memory locations. In this paper we introduce a model for parallel computation that is strictly weaker than the shared memory models. The model is based on an actual machine currently being constructed (ZMOB). We examine the communication facilities available in the model and show that lower bounds for merging and searching on shared memory models are attainable (within a constant). The main results reported in the paper are:

-an O(lgN/lgP) algorithm for searching an N-element sorted list distributed on P processors.
-an O(N/P) algorithm for merging two N-element lists on 2P processors.
-an O(lgn) algorithm for merging two N-element lists on 2N processors.
-criteria and techniques for simulating CREW PRAM algorithms on ZMOB. One of the techniques is used to establish an O(lglgN) lower bound for merging two N-element lists on 2N processors.

Research sponsored by the Air Force Office of Scientific Research (AFSC), under Contract F49620-83-C-0082. The United States Government is authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation hereon.

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): CAR-TR-64, CSC-TR-1405	5. MONITORING ORGANIZATION REPORT NUMBER(S): AFOSR-TR- 84-0716
6a. NAME OF PERFORMING ORGANIZATION: University of Maryland; 6b. OFFICE SYMBOL (If applicable):	7a. NAME OF MONITORING ORGANIZATION: Air Force Office of Scientific Research
6c. ADDRESS (City, State, and ZIP Code): Center for Automation Research, College Park MD 20742	7b. ADDRESS (City, State, and ZIP Code): Directorate of Mathematical & Information Sciences, AFOSR, Bolling AFB DC 20332
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: AFOSR; 8b. OFFICE SYMBOL (If applicable): NM	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: F49620-83-C-0082
8c. ADDRESS (City, State, and ZIP Code): Bolling AFB DC 20332	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. 61102F; PROJECT NO. 2304; TASK NO. A2; WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification): PARALLEL SEARCHING AND MERGING ON ZMOB.	
12. PERSONAL AUTHOR(S): Simon Kasif	
13a. TYPE OF REPORT: Technical; 13b. TIME COVERED: FROM _____ TO _____; 14. DATE OF REPORT (Year, Month, Day): June 1984; 15. PAGE COUNT: 38	
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Parallel processing; ZMOB; searching; merging.

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

One of the most difficult issues that must be addressed when studying a class of parallel algorithms is the problem of choosing a model that captures the inherent difficulty of implementing these algorithms on a multiprocessor architecture. Shared memory models have proven to be an effective tool for deriving lower bounds on the complexity of comparison problems. In particular, a speed-up of lg(P) is possible for the problem of finding an element in an N-element sorted list, and speed-ups of P/lqlqp and P are possible for merging N-element sorted lists on P processors for the cases N+P and P N respectively.

In practice, these speed-ups are not attainable since the shared memory models ignore many practical considerations in multiprocessor systems, such as interprocessor communications, distribution of data on local memories and limited fan-out of memory locations. In this paper we introduce a model for parallel computation that is strictly weaker than the shared memory models. The model is based on an actual machine currently being constructed (CONTINUED)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL: Dr. Robert N. Buchal	22b. TELEPHONE (Include Area Code): (202) 767- 4939; 22c. OFFICE SYMBOL: NM

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

ITEM #19, ABSTRACT, CONTINUED: (ZMOB). We examine the communication facilities available in the model and show that lower bounds for merging and searching on shared memory models are attainable (within a constant). The main results reported in the paper are:

- an O (lgN/lgP) algorithm for searching an N-element sorted list distributed on P processors;

- an O(N/P) algorithm for merging two N-element lists on 2P processors;

- an O(lgn) algorithm for merging two N-element lists on 2N processors;

- criteria and techniques for simulating CREW and PRAM algorithms on ZMOB. One of the techniques is used to establish an O(lgnlgN) lower bound for merging two N-element lists on 2N processors.

1. Introduction

During the last decade a significant amount of progress has been made towards the understanding of the value of parallelism for specific computational problems. Sorting, searching and merging are three of the most fundamental tasks in computer science. Their significance is due to the key role they play in many domains of application. With the emergence of VLSI technology it is inevitable for us to wonder how fast these tasks may be performed on a parallel (multiprocessor) machine. However, in contrast to Von-Neumann machines whose execution is well understood, and for which we have a well established theory of the time complexity of comparison problems, this is not the case for parallel machines. The problem is due to the difficulty in correctly modeling the execution of a physically realizable parallel computer. Thus, one of the most difficult issues that must be addressed when studying a class of parallel algorithms is the problem of chosing a model that captures the inherent difficulty of implementing these algorithms on a multiprocessor architecture.

Many models have been proposed to solve this problem. Roughly speaking, parallel models of computation belong to two categories: shared memory models and fixed interconnection models. A typical shared memory model allows many processors to read the same location simultaneously,

but disallows concurrent writes to the same location. Since the model allows concurrent reads and insists on exclusive writes it is known in the literature as the CREW PRAM. Examples of fixed interconnection networks are the shuffle exchange network, mesh-connected array and n-dimensional hypercube.

Shared memory models are currently not realizable in practice. However, they serve as powerful analysis tools to derive lower bounds for parallel computers. That is, if we can show that the worst case time complexity of an algorithm is O(N) for the CREW PRAM, we have also established that no fixed interconnection network based parallel machine can perform faster (see [1] for a spectrum of models and their relationships). Establishing lower bounds for searching, merging and sorting was indeed the motivation for the powerful comparison model introduced by Valiant. In [8] several optimal algorithms for comparison problems are presented. The algorithm for merging was later shown to be implementable on the CREW PRAM [1],[3].

This paper is a further step in this direction. We introduce a model for parallel computation, called ZMOB. The ZMOB model is shown to be strictly weaker than the CREW PRAM. Despite this fact we demonstrate that within the constraints of this model we are still able to achieve (up to a constant) the lower bounds for searching and

merging that are attainable on a more powerful model of computation. Additionally, we define two fundamental criteria, that if obeyed allow us to simulate any CREW PRAM algorithm on the model investigated in this paper.
The outline of the paper is as follows:

In Section 2 we describe the parallel model of computation used in this paper, and establish its relationship to the CREW PRAM.

In Section 3 we investigate the problem of searching an N-element sorted list. We define the criteria for optimality of distribution of elements to processors, and give an allocation function of elements to processors that allows us to search the list in O(N/P) time. This algorithm is shown to be optimal. Since no communication is needed after the element searched for is broadcast to all the processors the result in Section 3 is not restricted to ZMOB.

In Section 4 we present three algorithms for merging two N-element strings on a P-processor ZMOB. Two of the algorithms are shown to be optimal up to a constant.

A-1

Finally, in Section 5 we conclude with some thoughts on extensions of the research reported in this paper, and with a discussion of the cost effectiveness of the model described in Section 2.

2. A Model for Parallel Computation

The model described in this section is based on ZMOB, a parallel multi-microprocessor system under development at the University of Maryland [4]). ZMOB is to consist of 256 Z80A microprocessors connected to a host computer (VAX-11/780). Communication between machines is via a high speed, 48 bit wide, 257 stage shift register called the "conveyor belt". Each processor is connected to the conveyor belt via a collection of high speed 8-bit I/O registers, called its "mail stop". The registers are in charge of interrupt control, buffering and address control functions. The system is described in detail in [4,6]. We shall briefly describe here only the communication features necessary to understand the material in the sections that follow.

2.1. ZMOB communication facilities

As mentioned above the processors communicate by sending messages to each other on the " conveyor belt ". Each processor sends information using its own uniquely determined location on the belt, called its bin. Each

processor may read information from any bin, including its own, depending on the control bits set in the message contained in the bin. That is, each processor can theoretically consume any bin that is currently at its mail stop, but it can send information out only in its own bin. The control bits in the message allow the implementation of several communication strategies, as explained below. The message contained in each bin may be described by a 4-tuple: (C X S D), where C, X, S, D correspond to control, message content, source address, and destination address, respectively. Different control bits specify the following communication formats:

COMM-1.

Direct addressing - The message X is sent to a processor whose physical address is D.

COMM-2.

Pattern matching - Message X is sent to the first processor whose pattern (determined by Capability Code and Mask Registers in the Mail Stop) matches D.

COMM-3.

Send to all processors - Message X is sent to all processors.

COMM-4.

Send to a set of processors - Message X is sent to all processors whose patterns match D. Additionally,

different settings of Control Registers in the Mail Stop allow the following:

COMM-5.

Exclusive source - This mode provides exclusive conversation between two processors and disables interrupts from other processors.

COMM-6.

Readback - This mode allows an individual processor to intercept any of its own messages that has gone around the conveyor belt and was not consumed by any destination processor.

Though in principle ZMOB is an asynchronous machine, for the purpose of this paper we shall assume synchronous operation. That is, we assume that a unit time is a complete revolution of the belt, starting from the point when every bin resides at the mail stop of the processor that owns that bin. The unit time ends when each processor has had a chance to read the message that was sent to it. As we shall see shortly, at each communication step only one message may be sent to any processor. Moreover, at each communication step each processor may send out only one message.

2.2. ZMOB as a model for parallel computation

Formalizing the above concepts, at each execution step ZMOB may be modeled as a directed graph. The nodes in the graph correspond to the processors connected on the belt, and the arcs correspond to the communication links among the processors. Each processor is assumed to have internal memory. The hardware configuration allows for a processor to communicate with all the processors connected to it in one revolution of the belt. The belt is assumed to be so fast relative to the processors that each processor can communicate with the processors connected to it in unit time. The interconnection topology at each step is determined as follows:

1. If PE_i sends a message by physical address then depending whether it was sent by COMM-1 or COMM-3 it is assumed to be connected to one or all processors in the graph.

2. If PE_i sends a message by pattern α to a set of processors then it is assumed to be connected to all the processors that have α as their receiving pattern.

3. Each processor may have multiple outgoing arcs. However, it may output only one message at a time.

4. Each processor may have multiple incoming arcs. However, it may receive only one message at a time. To

prevent confusion all the algorithms presented in the paper assume that a processor may have at most one incoming arc.

5. A processor may not have incoming and outgoing arcs at the same time. That is, at each revolution of the belt the processor is either receiving or sending information, but not both.

To simplify matters we shall assume that each execution step of ZMOB consists of two phases: a communication step and an execution step. The communication step is further subdivided into a sending step, when all the processors load their respective bins, and a receiving step, when all the processors consume the messages sent to them during the sending step. The execution step is assumed to be one computational step of the processor, e.g., comparison and addition of two integers. The computational step is assumed to be at least as long as the communication step.

Conditions 1-3 make the model more powerful than a simple communication ring. Conditions 4-5 distinguish the model from shared memory models such as the CREW-PRAM. To see that ZMOB is indeed weaker than a model that allows P processors to read simultaneously from any location, one needs to envision a CREW PRAM algorithm that in one atomic step performs P reads from P locations, stored in a N-

location shared memory. To simulate the above situation on ZMOB, we need to distribute the N locations on P processors, by storing N/P locations in the internal memory of each processor. Now, in the worst case, all the simultaneously read locations may reside in the same PE. Consequently, if P processors are to read from some processor PE_i , we have P outgoing arcs from PE_i. However, by condition 4, PE_i can output only one element at a time. Thus, a P-processor ZMOB can simulate an arbitrary P-processor CREW PRAM in O(P) time, and the bound is tight.

In the following sections we shall show several nontrivial adaptations of CREW PRAM algorithms for ZMOB, sacrificing only a constant factor.

3. Searching

In this section we consider the problem of searching for an element in an N-element sorted list , distributed on P processors. The first thought that comes to mind is: distribute the N elements evenly among the P processors and perform a sequential binary search on each processor. However, this intuitively appealing approach results in a negligible speed-up since each processor performs $\lg\frac{N}{P}=\lg N-\lg P$ comparisons.

3.1. Algorithm SEARCH

We first present an algorithm for searching that may be implemented on a shared memory computer capable of performing concurrent reads.

Let $x=(x_1,\ldots,x_N)$ be a sorted list of N elements. For simplicity assume $N=s^{n+1}-1$, where $s=P+1$. Algorithm SEARCH works as follows:

0. Let $k=n$

1. Mark the elements of x subscripted by $is^k, 1\leq i$.

2. Assign PE_i to element is^k and compare it to the element being searched for. At the end of this step PE_i records the result of the comparison in location loc_i.

3. Now each PE_i compares loc_i , and loc_{i-1} , for $1<i$. If for some j $loc_j \neq loc_{j-1}$, then the searched element is in the j-th interval.

4 Let $k=k-1$ and reindex the elements of the j-th interval by l, $1\leq l\leq s^k$.

5. Repeat steps 1-5 for the elements in the j-th interval.

Since P comparisons are performed simultaneously in steps 2-3 we reduce the problem of searching s^k elements to a problem of searching $s^{(k-1)}$ elements. Thus, in Cn

steps we can search the entire list (for some constant C). Consequently the time complexity of the algorithm SEARCH is of order

$$n=\frac{\lg(N+1)}{\lg(P+1)}$$

Referring to [2] we note that the algorithm is optimal.

Note that step 3 in SEARCH may be performed on the ZMOB model introduced in Section 2 in one communication step, where PE_i sends loc_i to PE_{i+1}. Unfortunately, step 2 in the algorithm does not readily apply on the ZMOB model if we distribute the elements of x by assigning the first $\frac{N}{P}$ elements to the first PE, the following $\frac{N}{P}$ elements to the second, and so on. To see this, one needs to observe that after the first comparison <u>all</u> the elements that need to be compared in the next step reside in the same PE. Since the model allows us to output only one location at a time we have to multiply the complexity of the algorithm by P, yielding $P\lg\frac{N}{\lg P}$. Obviously, one can allocate all N elements of the list x to every PE; but this is hardly an optimal solution.

The analysis presented above suggested the question of whether there is an allocation of elements to processors that has the following desirable property:

<u>Definition</u>: Given a parallel algorithm A, an allocation of elements to the processors is said to be a <u>good alloca-</u>

tion for A if at any time during the execution of A whenever PE_i performs an operation on some element, that element already resides in the memory of PE_i .

Lemma 3.1:

Let $x=(x_1,x_2,\ldots,x_N)$ be a sorted string. We shall assume $N=s^{n+1}-1$, where $s=P+1$. Let s_j be the representation of j, $1\leq j\leq N$ using the base s, i.e.,

$$j=a_{j_n}s^n+a_{j_{n-1}}s^{n-1}+,\ldots,+a_{j_1}s^1+a_{j_0}$$

Then, in algorithm SEARCH if PE_i performs a comparison on the element x_j at step k then

$$j=a_{j_n}s^n+a_{j_{n-1}}s^{n-1}+,\ldots,+a_{j_{n-k+1}}s^{n-k+1}$$

and $a_{j_{n-k+1}}=i$.

Proof:

We shall prove the lemma using induction on the number of steps k.

For k=1 the elements being compared are indexed by is^n, $1\leq i$ and PE_i compares element is^n. Thus, if x_j is being compared by PE_i, then $s_j=is^n$, and $a_{j_n}=i$.

For $1<k$, assume PE_i compares x_j. By the induction hypothesis the elements compared at step k-1 were indexed by integers of the form

$$j=a_{j_n}s^n+a_{j_{n-1}}s^{n-1}+,\ldots,+rs^{n-(k-1)+1}$$

for $1\leq r\leq s-1$. Assuming that the next searched interval is given by

$$a_{j_n}s^n+a_{j_{n-1}}s^{n-1}+,\ldots,+(r_0-1)s^{n-k+2},a_{j_n}s^n+a_{j_{n-1}}s^{n-1}+,\ldots,+r_0s^{n-k+2}$$

we find that at step k PE_i compares the element subscripted by

$$a_{j_n}s^n+a_{j_{n-1}}s^{n-1}+,\ldots,+(r_0-1)s^{n-k+2}+is^{n-k+1}$$

Q.E.D.

As an immediate corollary of the lemma we have

Theorem 3.2:

Let x,s,N be as above. Let c_j be the coefficient of the term with the smallest exponent in s_j, i.e., the representation of j in the base s. Let f(j) be the allocation function of elements of x to processors PE_i, $1\leq i\leq P$ defined by

$$f(j)=c_j$$

Then f(j) is a good allocation function for algorithm SEARCH.

Proof:

The proof follows immediately from Lemma 3.1 by observing that whenever PE_i compares element j it must be

the case that

$$c_j = i$$

Q.E.D.

4. Merging on ZMOB

In this section we address the problem of merging two sorted strings of N numbers using P processors in a multiprocessor like ZMOB. We present three different algorithms for merging with the following characteristics:

1. All have a sublinear lower bound.
2. All are based on enumeration - that is, at the end of the merging every element knows its absolute location in the string of length 2N obtained by merging the two input strings of length N.

The algorithm for merging presented in Section 4.1 has a lower bound of lg N. It fully utilizes the pattern matching capabilities of ZMOB. The algorithm presented in Section 4.2 for merging two N-elements lists on 2N processors is a nontrivial adaptation of Kruskal's merging algorithm [3] for CREW-PRAM to ZMOB. This algorithm is optimal (up to a constant).

The two algorithms in Sections 4.1 - 4.2 do not generalize to the case of merging N-element lists on P

processors for P<N . Therefore, in Section 4.3 we present an optimal algorithm for merging two lists of length N each on 2P processors for the case P<N . The time complexity of this algorithm is of order $\frac{N}{P}$. The optimality of the algorithm in Section 4.3 is based on the observation that every parallel algorithm of time complexity T may be converted to a sequential algorithm of complexity PT and hence every parallel merging algorithm of order $O(\frac{N}{P})$ is optimal.

4.1. Merging Using Selective Broadcasting

In this section we present an algorithm for merging two N-element sorted strings of integers on a 2N-processor ZMOB. We present a fairly detailed algorithm in order to provide the user with intuition as to how a machine like ZMOB may be programmed.

Let $x=(x_1,\ldots,x_N)$ and $y=(y_1,\ldots,y_N)$ be two sorted lists of integers. The first string is stored in processors $P_1,\ldots,P_N$, subsequently referred to as the x-processors, and the second string is stored in processors $P_{N+1},\ldots,P_{2N}$, referred to as the y-processors. The elements are stored in ascending order, one element per processor. To simplify the discussion we shall assume without loss of generality that $N=2^n-1$, and that the end elements of the string y, y_1 and y_N, are -infinity and +infinity respectively. Each processor has the following

variables.

1. N[i] = the integer stored in PE_i
2. Index[i] = the position of N[i] in the string x or y depending whether PE_i is an x-processor or a y-processor.
3. PATTERN[i] = the pattern to be used by PE_i during the selective broadcast operation.
4. RES[i] = the result of the comparison performed by PE_i.
5. ENUM[i] = the final position of N[i] in an ordered enumeration of the strings x and y.
6. TEMP[i] = the location of element x_i in the string y. By location we mean the index j such that $y_j \leq x_i < y_{j+1}$.

For simplicity algorithm MERGE_4.1 is subdivided into two phases. During the first phase we call the procedure FIND_PART_ORDER, with the strings x and v, that for each element x_i finds its position in the list v. The positions are stored in TEMP[i]. In the second phase the procedure FIND_TOTAL_ORDER is called to find the absolute location of each element in the resulting string.

Procedure FIND_PART_ORDER is called with parameters (l_x,u_x) and (l_y,u_y) which correspond to the lower and upper bounds of the two sorted strings x and y to be

merged. For simplicity we assume that the elements in the strings are distinct. Initially we call FIND_PART_ORDER with the parameters 1,N,N+1 and 2N. Procedure FIND_PART_ORDER is given below.

Procedure FIND_PART_ORDER(l_x, u_x, l_y, u_y)

begin

$i = \frac{u_x - l_x}{2} + l_x$

0. For all j such that $l_y \leq j \leq u_y$ set PATTERN[j] = i.

1. Broadcast N[i] to all processors PE_j such that $l_y \leq j \leq u_y$.

2. For all j such that $l_y \leq j \leq u_y$ let PE_j compare N[j] to N[i] and store the result of the comparison in RES[J].

3. By comparing RES[j] with RES[j+1], for all j, such that $l_y \leq j \leq u_y$ find the location of N[i] in the string y and broadcast it to PE_i .

4. Let TEMP[i] = the location of N[i] in y.
Observe that $l_y - 1 \leq$ TEMP[i] $\leq u_y$.

if $l_x < i$ then

begin

5. For all j, such that $l_y \leq j \leq u_y$ set

$$\text{PATTERN}[j] = \begin{cases} (i-l_x)/2+l_x & \text{if } N[j] \leq \text{TEMP}[i] \\ (u_x-i)/2+i & \text{if } N[j] > \text{TEMP}[i] \end{cases}$$

6. Let

$ll_x = l_x$

$ul_x = i-1$

$l2_x = i+1$

$u2_x = u_x$

7. Let

$ll_y = l_y$

$ul_y = \text{TEMP}[i]$

$l2_y = \text{TEMP}[i]+1$

$u2_y = u_y$

cobegin step 8 and step 9.

8. If $ll_y > ul_y$ then

For all i, $ll_x \leq i \leq ul_x$, set TEMP[i] to ul_y.

If $ll_y = ul_y$ then

begin

FIND_PART_ORDER(ll_y, ul_y, ll_x, ul_x)

For all i, $ll_x \leq i \leq ul_x$, such that $i \leq \text{TEMP}[ll_y]$ set

TEMP[i] = $ll_y - 1$.

For all i, $ll_x \leq i \leq ul_x$, such that $i > \text{TEMP}[ll_y]$ set

TEMP[i] = ll_y

```
    end
    Otherwise, ( ll_y<ul_y ) call
      FIND_PART_ORDER( ll_x,ul_x,ll_y,ul_y )

    9. If l2_y>u2_y then
    For all i, l2_x≤i≤u2_x, set TEMP[i] to u2_y
    If l2_y=u2_y then
    begin
      FIND_PART_ORDER( l2_y,u2_y,ll_x,ul_x )
    For all i, l2_x≤i≤u2_x, such that i≤TEMP[l2_y] set
      TEMP[i] = l2_y-1
    For all i, l2_x≤i≤u2_x, such that i≥TEMP[l2_y] set
      TEMP[i] = l2_y
    end
    Otherwise, ( l2_y<u2_y ) call
      FIND_PART_ORDER( l2_x,u2_x,l2_y,l2_y )

    coend steps 8-9.

  end

end
```

Explanation:

Intuitively, procedure FIND_PART_ORDER works as follows:

At steps 0-1 the middle element of x, x_i, is broadcast to all the y-processors. This defines two segments of x, X_1 and X_2. In step 6 we determine the lower and the upper bounds of each X-segment. Since we conveniently have chosen N to be 2^n-1, the length of each x-segment is $\frac{N+1}{2}-1$.

At step 3 we find the location of the element in v. This defines two segments of y, Y_1 and Y_2. In step 7 we determine the lower and the upper bounds of each y-segment.

Clearly we can now separately merge X_1 with Y_1 and X_2 with Y_2. This is accomplished with the recursive calls in steps 6-8, and the set up of the y-processors in step 3. In step 3 all the Y_1 processors set their patterns to the index of the middle element of X_1. Similarly the X_2 processors set their pattern to the middle element of X_2.

There are several cases that need to be taken care of separately. The first two cases arise when TEMP[i] < l_y, i.e. the middle element of x is smaller than all the elements in v, or when TEMP[i] = u sub v, i.e., the middle element of x is larger than all the elements in y. In the former case we merge all the elements of X_1 to the left of l_y, and the latter case we merge all the elements of X_2 to the right of u_y.

The last exceptional case that needs to be taken care of is the case when either Y_1 or Y_2 is of length 1. Without loss of generality assume that $|Y_1|=1$, the element in Y_1 is y_I and the integer in Y_1 is N[I] . Clearly, in this case the elements of X_1 will be merged either to the left or to the right of the element in Y_1 . Thus, the position of each element in the segment X_1 may be determined by inserting the singleton element of y in X_1, which may be done by calling FIND_PART_ORDER with Y_1 and X_1 in this order. Once the location of the y element in X_1 has been determined we can set the location of all the elements in X_1 greater than N[I] to be N[i], and the locations of all the elements in X_1 smaller then N[1] to be N[i]-1 .

Procedure FIND_PART_ORDER terminates when the size of all x-segments is zero.

Proposition 4.1:

Algorithm FIND_PART_ORDER correctly finds the relative location of the x-elements in the string y in O(lgN) time.

Proof:

It is easy to see that each invocation of FIND_PART_ORDER with segments X , Y either finds the location of X in Y in case X is a singleton list, or

creates two recursive calls: FIND_PART_ORDER with (X_1 , Y_1) and FIND_PART_ORDER with (X_2 , Y_2). The lengths of X_1 and X_2 are smaller than $\frac{|X|}{2}$. Thus, algorithm FIND_PART_ORDER is guaranteed to terminate. It is also easy to verify that algorithm FIND_PART_ORDER spans a binary tree of recursive calls to FIND_PART_ORDER. At level k of the binary tree 2^k elements of X are merged into Y simultaneously. At the root level the middle element of X is compared to all the elements of Y, at the second level two elements of X are compared to the "right" elements of Y, and so on. Consequently, it suffices to prove by induction that at each level the chosen elements of X are merged into the string Y in the correct locations. The proof using induction is similar to the proof for Lemma 3.1 and is left as an exercise to the reader.

We must ensure that on each level, each processor is required to compare its local element to only one element of the other string. However, a processor may be asked to compare more than one element at a time only if two calls to FIND_PART_ORDER are created with the same strings, and this may happen only if the procedure FIND_PART_ORDER is called with a Y or X string of length one. This case is taken care of by the special check in steps 7-8.

Thus, algorithm FIND_PART_ORDER correctly finds the partial order of the elements of X in Y in O(lg(N)) time.

At this point each x-processor knows its relative location in the y-string. Obviouslv, we can reverse the parameters in the call to FIND_PART_ORDER and determine the relative location of each y-element in x. Now, it remains to show that we can find the absolute location of each element; this is done by procedure FIND_TOTAL_ORDER given below.

The input to FIND_TOTAL_ORDER is two sorted strings x and y, such that $|x|=|y|=N$. We assume each element in x knows its relative location in y. We recall that by the relative location of x_i in y we mean the j, $1 \leq j \leq N$ such that $y_j \leq x_i < y_{j+1}$. We also recall that each relative location is stored in TEMP[i].

The output of FIND_TOTAL_ORDER is: each x-processor knows its absolute location in the string resulting from merging x and y.

Algorithm FIND_TOTAL_ORDER (x,v)

begin

1. For each j, $1 \leq j \leq N-1$ check if TEMP[j] < TEMP[j + 1]. Mark all those PEs whose TEMP[j] is greater than TEMP[j-1]. We shall refer to such a processor as a locally minimal PE.
2. For each j, $2 \leq j \leq N$ check if TEMP[j] > TEMP[j - 1]. Mark all those PEs whose TEMP[j] is smaller than TEMP[j + 1]. We shall refer to such a processor as a locally

maximal PE.

3. Now let each locally maximal x-processor PE_j send its index INDEX[j] by pattern to the y-processor indexed by TEMP[j] + 1.

At the end of this step each y-processor that received a message from an x-processor can compute its absolute position by adding the message content to its own index. Note that each y-processor may receive at most one message.

4. Now, for all j $1 \leq j \leq N$, such that PE_j is not a locally minimal PE, have PE_j do:

```
PATTERN[j] = TEMP[j]
```

5. Now, for all j $1 \leq j \leq N$, such that PE_j is the locally minimal PE, have PE_j do:

```
Send INDEX[j] by pattern TEMP[j]
```

At the end of this step each PE_j may compute its relative position among all those PEs with the same relative location in y. This may be done by subtracting the index of the locally minimal PE from the index of the processor.

6. Now, for all j $1 \leq j \leq N$ have PE_j do:

```
PATTERN[j] = TEMP[j]
```

7. Let all the y-processors PE_k that know their absolute location broadcast it by the pattern INDEX[k] - 1.

8. Now each x-processor may compute its absolute location by adding the absolute location of y-processor

received at step 7 and the relative position computed at step 5.

end

Notes:

1. All the steps in FIND_TOTAL_ORDER are atomic steps. Thus, the time complexity of FIND_TOTAL_ORDER is constant.

2. By reversing parameters in the call to FIND_TOTAL_ORDER we may find the absolute locations of the elements of y in x.

Theorem 4.1:

Let x and y be two sorted strings of length N, distributed in ascending order on a 2N-processor ZMOB. Then we can sort the two strings in O(lgN) time.

Proof: The theorem follows from Proposition 4.1 and the constant time complexity of procedure FIND_TOTAL_ORDER.

4.2. Optimal Merging on ZMOB

In Section 3 we found that one way to achieve lower bound performance on ZMOB is by distributing the information in such a way that each time the P processors perform P reads, we insure that each of the elements accessed by PE_j already resides in the memory of PE_j. However, this

is not always possible. In this section we use one more fundamental trick that allows deriving an optimal (up to a constant) merging algorithm.

Definition:

Given a parallel algorithm A, an allocation of elements to the processors is said to be a nearly-good allocation for A if at any time during the execution of A when two distinct PE_is perform an operation on two elements s and r the following holds:

1. The elements r and s reside in two different processors.
2. If the elements reside in the same PE then s = r.

Lemma 4.2:

Let A be some CREW PRAM algorithm and let f be a nearly-good allocation function for A. Then the algorithm A may be simulated on ZMOB in constant time.

Proof:

Each time P processors want to read P locations, they broadcast the requests for these locations. We shall assume that each processor knows to what PE the location it is trying to read has been allocated. Thus, at each concurrent read step of algorithm A, P requests for locations are broadcast on the bus.

Clearly, if there is no contention for processors the fetches may be performed by having each PE that has the required location respond to the sender.

A problem may occur in cases when two or more processors are contending for locations in the same processor. However, if f was a nearly-good allocation function for A, all the requested elements that reside in the same PE are equal. Thus, the processor that received the request for a location may send the location out by using its own index as a pattern. Irrespective of the number of requests sent to a processor it will consume and respond to one request only. If all the processors that requested the location set their receiving pattern to the index of the processor they requested information from, the data will be delivered to all those processors in unit time. Thus, algorithm A may be simulated on ZMOB in constant time.

Q.E.D.

The lemma above has an immediate corollary.

Corollary 4.2:

A P-processor P-element memory CREW-PRAM algorithm may be simulated on ZMOB in constant time.

Corollary 4.2 has an immediate important application to merging. In [3] we find an algorithm for merging two

sorted N-element strings on an N-processor CREW-PRAM in O(lglg(N)) time. The algorithm is optimal up to a constant [8]. Therefore, by Corollary 4.2 the algorithm may be simulated on a 2N-processor ZMOB with constant time overhead. Kruskal's algorithm performs the same function procedure FIND_PART_ORDER performs in Section 4.2, that is, for each element it finds its relative location in the other string. Recalling that the time complexity of FIND_TOTAL_ORDER for N=P is constant we conclude:

Theorem 4.2:

The lower bound for merging two N-element sorted strings on a 2N-processor ZMOB is O(lglg(N)).

4.3. Merging two long strings with a small number of processors

Let X Y be two sorted strings, and assume $|X|=|Y|=N$. In this section we show that we can merge the two strings on a 2P processor ZMOB in O(N/P) time using algorithm MERGE_4.3.

The input to MERGE_4.3 is given in the form of two strings that are initially distributed in ascending order on 2P processors. For simplicity assume N=SP.
The output of MERGE_4.3 is: each processor knows its absolute location in the string resulting from the merge. Algorithm MERGE_4.3 is given below.

Algorithm MERGE_4.3

1. Choose the elements indexed by iS in each string . There are no more than P chosen elements in each string. Moreover, there is only one chosen element in each processor. In fact the chosen element is the last element in each PE.

2. Merge the chosen elements of each the strings. This step may be done in O(lglgP) time as shown in Section 4.2. Now, note that the chosen elements define P equal-length intervals in each string, denoted by $(X_1,\ldots,X_P)$ and $(Y_1,\ldots,Y_P)$. At the end of this step we know to what interval in the other string each chosen element belongs.

In step 3 we find the exact relative position of each chosen element of X in Y. Delegate an x-processor to each chosen element of X. Set the pattern of this processor to be the index of the interval in Y that the chosen element belongs to. Formally, let PE_{iS} set its receiving pattern to the index of the y-interval that element x_{iS} is in.

3. Broadcast the content of each y-interval on the belt. Note that each x-processor communicates with only one y-processor, while a y-processor may be communicating with more than one x-processor or none at all. At the end of this step each x-processor contains the content of the y-interval where the chosen element of x, residing in the x-processor, belongs. Now each x-processor may find its relative position in the string y, by performing a

sequential binary search on the content of the y-interval it contains. Since the length of each y-interval is bounded by $\frac{N}{P}$, and each processor is required to output or receive only one element at a time, the time complexity of steps 3 is of order $O(N/P) + lg(N/P)$.

4. Repeat the above step for the chosen elements of Y. The positions of the elements chosen in step 1 and the relative positions found in steps 3-4 define 2P segments in each string denoted by X_{i_j} and Y_{i_j} respectively, where $1 \leq j \leq 2P$. Each segment is defined uniquely by its right end-point. This subdivision defines 2P disjoint pairs (X_{i_j}, Y_{i_j}) that may be merged separately.

5. Now, we DELEGATE (see below) 2P processors to the 2P pairs. Once the pair of x-y segments indexed by the same integer i_j, $1 \leq i_j \leq 2P$, reside in the chosen processor, we may merge them sequentially in $O(\frac{N}{P})$ time. This is true since each segment in each pair is at most $(\frac{N}{P})$ long.

We must, therefore, show that DELEGATING 2P processors to the corresponding pairs of x-y segments may be accomplished in $O(\frac{N}{P})$ time. Without loss of generality we shall discuss only the x-segments.

We first observe that at the end of step 4 each of the 2P processors PE_j contains an integer e_j, $0 \leq e_j \leq N$, that corresponds to the end of some x-segment. The x-processors contain the ends of the x-segments defined in

step 1, while the y-processors contain the ends of the x-segments created in step 4. Additionally, each of the 2P processors contains some interval X_i, which is the superset of the x-segment defined uniquely by the integer e_j. Thus, each PEj must determine the index i_j and the lower bound of the segment defined by e_j. This task is performed in O(N/P) steps by Algorithm 4.4. Once this problem is solved we may simply DELEGATE the i_jth segment to the processor in which the end of that segment resides.

Algorithm 4.4

1. Note that the integers in the x-processors define a sorted list. Also note that the integers in the y-processors define a sorted list. Thus, we may find the absolute order simply by merging the integers in the x-processors with the integers in the y-processors. This may be done in O(lglgP) time. At the end of step 1 each processor knows the index of the segment it is responsible for and its upper bound.

2. Let T[j] be the index of PE_j in the new enumeration. In two steps PE_j can find the lower bound of the segment stored in PE_j by communicating to $PE_{T[j]-1}$. This is done by letting each PE_j set its pattern to T[j], and then letting each PE_j send its T[j] by pattern to $PE_{T[j]+1}$.

Thus, the overall time complexity of Algorithm 4.4 is of order lglgP.

Similar arguments hold for the y-processors.

Finally, each of the 2P processors contains a sorted string of length at most 2N/P. The enumeration of all the 2N elements is straightforward and is left as an exercise to the reader. As a result we have the following theorem.

Theorem 4.3:

Let X, Y be two sorted strings, and asssume $|X|=|Y|=N$. For N = SP, $1 \leq S$ we can merge the two strings distributed in ascending order on a 2P-processor ZMOB in O(N/P) time using 2P processors.

Note:

Since the lower bound to merge two such strings on a sequential computer is of order O(N), we conclude that the algorithm presented above is optimal up to a constant.

5. Summary

In this paper we have investigated the problem of searching and merging two N-element strings on a parallel model of computation (ZMOB). The main results reported in this paper are:

1. Parallel searching for an element on a N-element string distributed on P processors may be performed

on ZMOB in O(N/P) time. The algorithm given is optimal up to a constant.

2. Two sorted strings of length N may be merged on a N-processor ZMOB in O(lglgN) time.

3. Two sorted strings of length N may be merged on an N-processor ZMOB in O(N/P) time.

The results reported in this paper may be used for solving a variety of problems. Clearly, the lgN merging algorithm of Section 4.1 may be used to derive a lg^2N sorting algorithm. In a subsequent paper we plan to extend the results in Section 4 to derive an optimal parallel algorithm for AND-ing (OR-ing) two binary strings represented by run length codes. In this representation a binary string is represented by the value of the first element of the string followed by a string of integers that represent the successive runs of 0s and 1s by their respective lengths. For a large class of binary strings this representation is more compact, and it is widely used in the domain of signal and image processing. AND-ing and OR-ing on a sequential computer may be performed in O(K) time using this representation, where K is the number of runs. We show that we can AND two strings having K runs each in O(K/P) time on P processors. Parallel processing of run length codes has not, to our knowledge, been previously studied.

Although the algorithms in this paper were described for a particular machine organization, they generalize naturally to any machine that satisfies conditions 1-5 in Section 2. Note that all the algorithms presented in Sections 3-6 used relatively few communication links among processors. In fact at any step of the execution the interconnection graphs induced by the algorithms had only P edges at the most. This is true since in all the algorithms either the x-processors communicated to the y-processors and vice versa or the x-y processors communicated among themselves in a mesh-connected fashion.

In [7] it was argued that broadcasting "cannot significantly aid sorting algorithms". In this paper we have demonstrated that selective broadcasting allows a dynamically reconfigurable interconnection topology, and proves to be an extremely powerful construct for mesh-connected like computers. In particular it allows one to attain maximal speed-ups for searching and merging.

Finally, we would like to conclude with a pragmatic remark. The philosophy behind ZMOB is simple. In many cases it is cost-effective to build a parallel machine from a collection of slow, cheap processors connected by a very fast communication medium. In other cases we would like to experiment with shared memory or dynamically reconfigurable systems. In either case ZMOB may be an

effective simulation tool. As mentioned in the introduction ZMOB is currently under construction at the University of Maryland. A 16-processor ZMOB is partially operational and is undergoing extensive testing and debugging, and other processors are being connected to the conveyor belt. The communication facilities described in Section 2 are fully supported on the belt though it is premature to conjecture whether the theoretical model introduced in this paper is indeed realizable in practice.

ACKNOWLEDGEMENTS

I thank Dr. Azriel Rosenfeld, Dr. Angela Wu and Mr. S.K. Bhaskar for numerous discussions and comments that contributed greatly to the development of the ideas in this paper and to the presentation of the results.

REFERENCES

[1] A. Borodin and J.E. Hopcroft, "Routing, Merging and Sorting on Parallel Models of Computation", Proc. ACM 14th Annual Symposium on Theory of Computing, 1982, p. 338-344.

[2] D.E. Knuth, The Art of Computer Programming, Sorting and Searching, Vol. 3, Addison-Wesley, Reading, Mass. 1973, p. 422.

[3] C.P. Kruskal, "Searching, Merging and Sorting in Parallel Computation", IEEE Transactions on Computers, Vol. C-32, 1983, pp. 942-946.

[4] C. Rieger, J. Bane and R. Trigg, "ZMOB: A Highly Parallel Multiprocessor", TR-911, Department of Computer Science, University of Maryland, College Park, May 1980.

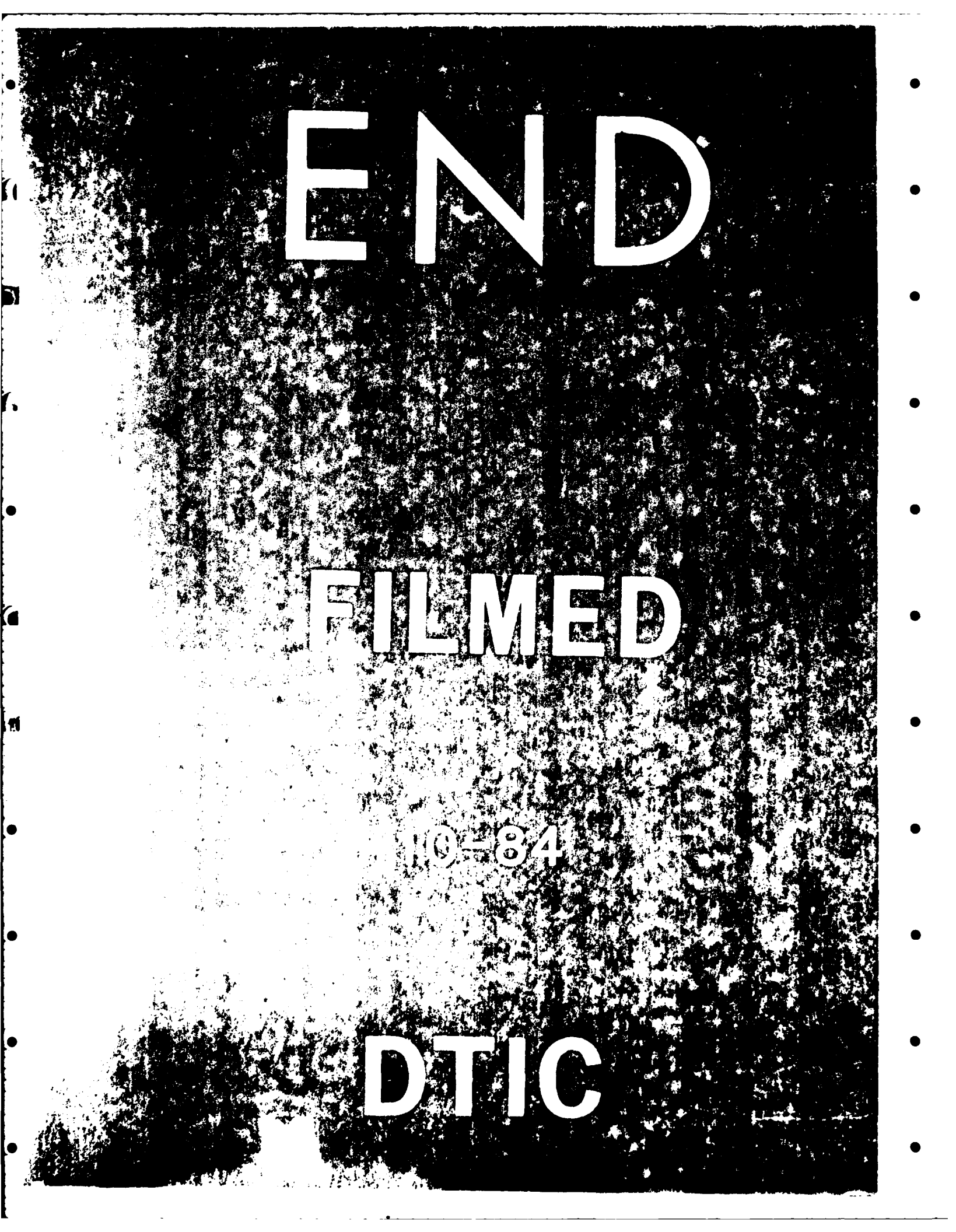
[5] C. Rieger, J. Bane and R. Trigg, "ZMOB: A New Computing Engine for AI", TR-1028, Department of Computer Science, University of Maryland, College Park, March 1981.

[6] C. Rieger, "ZMOB: Hardware from a User's Viewpoint", TR-1042, Department of Computer Science, University of Maryland, College Park, April 1981.

[7] Q.F. Stout, "Broadcasting in Mesh-Connected Computers", Proc. of the 1982 Conf. on Information Sciences

and Systems, Princeton University, Princeton, NJ., 1982, pp. 85-90.

[8] L.G. Valiant, "Parallelism in Comparison Problems", SIAM J. of Computing, Vol. 4, 1975, pp. 348-355.


END
FILMED
10-84
DTIC